

**Amendments to the drawings,**

*There are no amendments to the Drawings.*

RECEIVED  
CENTRAL FAX CENTER

SEP 13 2006

**Remarks**

Status of application

Claims 1-42 were examined and stand rejected in view of prior art. Claims 43-62 were withdrawn pursuant to a Restriction Requirement; these claims are now canceled. The pending claims have been amended to further clarify Applicant's invention. Reexamination and reconsideration are respectfully requested.

The invention

System and method for cloning of prepared statements for execution against a database are described. In one embodiment, for example, a method of the present invention is described for executing a database statement, the method comprises steps of: preparing at least one template capable of generating an executable statement for execution against a database via a particular database connection; storing the at least one template in a shared cache at an application server so that it is available to a plurality of database connections, such that memory requirements of the shared cache are proportional to how many templates exist, regardless of how many executable statements are generated; in response to a request to execute a particular statement on a given database connection, determining whether a template for the particular statement is available in the shared cache; if the template is available in the shared cache, creating a corresponding executable statement based on the template, the executable statement having been prepared for execution on the given database connection; and executing the corresponding executable statement on the given database connection.

General

A. Sec. 101 rejection

Claim 22 [23] stands rejected under Section 101, as being directed to nonstatutory subject matter, on the basis that the claim recites "a downloadable set of processor-executable instructions" that a software or program is being processed without any links to a practical result in the technology arts. The claim (which is claim 23, not 22) has been amended to couch the claim limitation in terms of a process step, thereby overcoming the rejection.

### B. Claim objection

Claim 7 is objected to on the basis that it contains the following informalities: the claimed limitation "the database client includes application server" is incorrect according to the specification that the database client process is application server (Fig. 4). The claims have been amended to cure this informality.

### Prior art rejections

#### A. First Section 103 rejection: Bireley and Bird

Claims 1-3, 9-14, 20-25, 16-19, 26, 32-35 and 37-42 stand rejected under 35 U.S.C. 103(a) as being unpatentable over Bireley et al (or hereinafter "Bireley") (US 6115703) in view of Bird et al (or hereinafter "Bird") (US 6598058). The Examiner's rejection of claim 1 is representative:

As to claim 1, Bireley teaches the claimed limitation a method for executing a database statement (col. 1, lines 48-65):

"preparing at least one template for execution of a statement against a database" as building an executable form or structure of the SQL statement for executing of the SQL statement against a database in DBMS. The executable form or structure is represented as a template (col. 1, lines 50-60);

"storing said at least one template in a shared cache available to a plurality of database connections" as saving the executable structures of the most recently executed SQL statements in a global cache. If an application program request a prepare for a SQL statement that has been globally cached, the entire preparation process can be skipped by obtaining a copy of the executable structures from the global cache. The global cache is available to database user applications as database connections (col. 1, lines 50-67; col. 2, lines 1-20; col. 4, lines 20-40);

"in response to a request to execute a particular statement on a given database connection, determining whether a template for said particular statement is available in the shared cache" as using a global cache area to save the executable structures of the most recently executed SQL statements. If an application program requests a prepare for a SQL statement that has been globally cached. For example, a second statement is received for execution from the application. It is determined that the second statement can be executed using the stored executable structure in cache, then the second statement is executed using the stored executable structure. The above information implies that the stored executable

structure is available for executing the second statement (col. 1, lines 65-67; col. 2, lines 1-5; col. 2, lines 33-40);

"and executing the database statement on the given database connection" as (col. 1, lines 50-67; col. 2, lines 1-20);

Bireley does not explicitly teach the claimed limitation "if the template is available in the shared cache, creating a database statement based on the template for execution on the given database connection". Bird teaches the statement portion 41 of dynamic cache 40 is used to support application requests to prepare the dynamic SQL statement 64 and obtain an executable section. The above information indicates that the statement portion 41 is available for creating the dynamic SQL statement (fig. 2, col. 8, lines 37-41).

It would have been obvious to a person of an ordinary skill in the art at the time the invention was made to apply Bird's teaching of the statement portion 41 of dynamic cache 40 is used to support application requests to prepare the dynamic SQL statement 64 and obtain an executable section to Bireley's system in order to avoid repeated compilation of identical SQL request so that given the potentially low cost of compilation.

For the reasons discussed below, Applicant's claimed invention is distinguishable on a number of grounds.

Bireley describes a technique for improving performance of a relational database system in a two-tier configuration. In that configuration, an application on the client computer connects to a database server and sends it a query. The database server, in turn, analyzes the query, and creates a binary structure (e.g., "plan") that can be cached at the server for future use, so that the query may execute more quickly in the future should the same type of query occur again. The cache is maintained at the database server so that upon subsequent execution of the same query the database server may find the plan in the cache. In this manner, the database server may execute the query more quickly, as it need not expend effort re-creating or building the plan. A query "plan" itself is a plan of how to execute the query efficiently. A "prepared statement," on the other hand, is an object that allows one to execute a query. More precisely, a "prepared statement" is a "reference" to a plan that exists in the database. Bird (at best) simply extends the Bireley's idea to run on a database cluster (i.e., multiple database processes sharing the database server-side cache in memory).

To be sure, both Applicant's approach and Bireley/Bird approach pertain to

improving performance of relational database systems, and out of necessity they will of course have many features in common. As described below, however, Applicant's approach is fundamentally different than that described by Bireley and/or Bird.

Applicant's invention provides an approach for improving performance of a relational database system in a three-tier configuration, which includes an application server running in a middle tier. The application server will typically execute the same query many more times than would have occurred in a two-tier configuration, as the application server is servicing a multitude of clients simultaneously. In a three-tier configuration, therefore, it is much more common for a given query to be repeatedly executed.

As a general proposition in architectures employing a middle tier, one faces a trade-off between memory usage and speed (CPU utilization) when determining whether or not a particular piece of data should be cached. The more items that are cached, the greater reduction in CPU requirements or utilization, however that occurs at the expense of increasingly larger amounts of required cache memory. Consider, for example, a scenario where a certain set of queries is executed over and over again. In order to improve performance, the queries are cached at the application server. Therefore, in this scenario, a given query template is stored in the cache so that it may be executed multiple times. (A "template" in this context is not referring to a query "plan.")

However, the cache requirement in this scenario quickly adds up, as each query in the cache must be bound to a particular connection to the database server. Upon retrieval from the cache, the template must be converted into a "prepared" query or execution statement (i.e., an executable query), which requires (among other things) that the statement be attached to a specific connection. The application server typically employs a pool of connections. Each connection is essentially an identical copy (i.e., the same connections to the same servers), duplicating socket information so that more than one query may be executed at a time. As the number of clients increases, the cache size requirement increases in a concomitant manner as an increasing number of connections from the connection pool (up to a maximum) are used to service the clients. The cache size ultimately required (i.e., required memory), therefore, is a function of the number of connections employed (simultaneously) multiplied by the number of query templates. In

other words, the Cartesian product of connections multiply by queries reflects the complexity of the in-memory storage required for the application server's cached execution statements.

Consider the following example. Suppose one had 10 queries that are normally run from the application server (i.e., 10 cached query templates), with 20 connections available from the connection pool. Eventually, when the application server is running at full capacity, 200 executable ("prepared") query statements are generated. Each statement is essentially the same statement copied 20 times, with a particular statement being associated with a particular connection in the pool. Applicant's invention allows one to have an application server-side template that corresponds to a single statement. Here, that template can "migrate" across the different connections. Importantly, at any given point in time, one only has as many statements in memory as are actually executing at the moment.

In order to understand the importance of Applicant's approach, consider the memory requirements in an application server prior to application of Applicant's invention. For  $n$  distinct queries that are going to be executed and  $m$  different connections in the connection pool, the amount of memory consumed is proportional to  $n$  multiply by  $m$ . If either one gets big, the system quickly exhausts cache space. Scalability stalls at that point: once the cache space is exhausted further increases in performance cannot be realized.

With Applicant's invention, the memory consumed is proportional only to  $n$  itself, that is, the number of distinct queries. Therefore, a template (of a "prepared statement") is created for each one of those, which is then stored in the application server cache. Each template itself is essentially fixed (i.e., non-changing) in the sense that the template itself need not be bound to a particular connection. In other words, a given template as it is stored in the application server cache is stored as a reusable query template, not as a final executable (prepared) query statement (i.e., that is permanently bound to a particular connection). Of course at run time, an executable (prepared) statement must eventually be derived from the underlying cached template (that it is based on). In accordance with Applicant's invention, however, the amount of time needed to essentially instantiate or "clone" an executable statement from a given template is trivial (i.e., executes fast). At

any given time, given  $m$  connections (i.e., it is possible to have as many as  $m$  statements executing), the total cache memory requirement is merely a function of  $m + n$ , not  $m \times n$ . Therefore, Applicant's approach executes quickly, and at the same time substantially conserves cache memory usage. By storing reusable query templates at the middle tier application server (which is not the same thing as storing "query plans" at the database server), coupled with Applicant's approach of rapid instantiation of executable statements from underlying templates, Applicant's approach provides a three-tier database system with substantially improved scalability and performance.

Applicant's independent claims have been amended to bring these distinctions to the forefront. For example, claim 1 has been amended to include claim limitations of (shown in amended form):

preparing at least one template ~~for~~ capable of generating an executable statement for execution of a statement against a database via a particular database connection;

storing said at least one template in a shared cache at an application server so that it is available to a plurality of database connections, such that memory requirements of said shared cache are proportional to how many templates exist, regardless of how many executable statements are generated;

As shown, the amendment clarifies that Applicant's cached templates are not simply cached prepared or executable statements. Instead, each template itself is essentially the fixed or non-changing portion of the query -- importantly, the template itself is not permanently bound to a particular connection. In this manner, templates may be stored in the shared cache at the application server "such that memory requirements of [the] shared cache are proportional to how many templates exist, regardless of how many executable statements are generated." Thus, at any given time, given  $m$  connections (i.e., it is possible to have as many as  $m$  statements executing), the total cache memory requirement is merely a function of  $m + n$ , not  $m \times n$ .

Additionally, the amended claim includes the claim limitations:

if the template is available in the shared cache, creating a corresponding executable database statement based on the template, the executable statement having been prepared for execution on the given database connection;

As shown, the claim now includes specific language showing how the template is used to instantiate or "clone" a corresponding executable statement: a corresponding executable statement is created based on the template, the executable statement having been prepared for execution on the given database connection. As described by Applicant's specification, Applicant's preferred embodiment has been implemented in a manner where this creation or "cloning" step occurs rapidly (i.e., at high speeds).

Even if one were to believe that caching itself were obvious at the middle tier, Applicant's claimed invention is not simply drawn to middle tier caching of query plans but, instead, to a technique that creates a special type of **template** (i.e., a template for instantiating "prepared statements"), one that can serve as an execution or prepared statement "factory" at run time. This is not the same as simply caching query plans (for example, in the manner of Bireley/Bird). The "prepared statement" generated by the application server at run time (from the "prepared statements" template) in turn allows one to execute a query for which there is a "query plan" in the database server. The query plan itself may or may not be cached, depending on the database server.

All told, Applicant's invention provides a solution that operates both at high speeds and operates with significantly improved cache memory utilization, thereby providing a three-tier database system with improved scalability and performance over prior art solutions. It is respectfully submitted that Bireley and Bird, whether taken alone or in combination, do not teach or suggest these features of Applicant's invention. Especially in light of the amendments to the base claims (as well as clarifying marks made above), it is believed that the claims distinguish over the art of record and that any rejection under Section 103 is overcome.



**B. Second Section 103 rejection: Bireley, Bird, and Saha**

Claims 4-7 and 27-30 stand rejected under 35 U.S.C. 103(a) as being unpatentable over Bireley (above) in view of Bird (above) and further in view of Saha et al ("Saha") (US 2003/0236780). Here, the Examiner essentially repeats his rejection under the combination of Bireley and Bird, but adds Saha for the proposition that it teaches Applicant's application server claim elements. The claims are believed to be allowable for at least the reasons cited above pertaining to the first rejection under Section 103, based on the combination of Bireley and Bird (above). In particular, neither Bireley nor Bird teaches or suggests Applicant's claimed approach which includes (among other things) caching templates (not query plans) at the middle tier application server, such that those templates can be used to spawn runtime execution statements (i.e., statements prepared for execution on a particular database connection). Nothing in Saha cures this deficiency of Bireley and Bird. Accordingly, the claims are believed to be allowable over the combination of all three references.

Additionally, the claims are believed to be allowable for the following additional reasons. Even if one were to add an application server to the combination of Bireley and Bird, one at best would simply have cached query plans at the application server (which may themselves be of little use, as these are database server-side objects), or simply have cached execution statements (i.e., objects tied to particular database connections). The putative combination of all three would create a system where the middle tier cache memory requirements are directly proportional to number of queries times number of connections. Clearly, this does not re-create Applicant's invention, where the middle tier cache memory requirements are directly proportional only to the number of queries. Accordingly, it is respectfully submitted that the combined references do not teach or suggest all of the elements present in Applicant's claimed invention, and that the references therefore do not establish obviousness under Section 103.

**C. Third Section 103 rejection: Bireley, Bird, and AAPA**

Claims 4-8, 15, 27-31 and 36 stand rejected under 35 U.S.C. 103(a) as being unpatentable over Bireley and Bird (above) and further in view of Applicant's admitted prior art (AAPA). Here, the Examiner essentially repeats his rejection under the

combination of Bireley and Bird, but adds AAPA for teaching, for example, the claim limitation of "the request to execute a particular statement on a given database connection is received at an application server." The claims are believed to be allowable for at least the reasons cited above pertaining to the first rejection under Section 103, based on the combination of Bireley and Bird (above). In particular, neither Bireley nor Bird teaches or suggests Applicant's claimed approach (as described in detail above). Nothing in the AAPA cures this deficiency of Bireley and Bird. The claims are believed to be allowable over the combination of all three references (especially in view of amendments to the base claims), thus overcoming any rejection under Section 103.

Any dependent claims not explicitly discussed are believed to be allowable by virtue of dependency from Applicant's independent claims, as discussed in detail above.

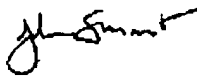
#### Conclusion

In view of the foregoing remarks and the amendment to the claims, it is believed that all claims are now in condition for allowance. Hence, it is respectfully requested that the application be passed to issue at an early date.

If for any reason the Examiner feels that a telephone conference would in any way expedite prosecution of the subject application, the Examiner is invited to telephone the undersigned at 408 884 1507.

Respectfully submitted,

Date: September 13, 2006

✓  Digitally signed by John A. Smart  
Date: 2006.09.13 18:14:21 -07'00'

John A. Smart; Reg. No. 34,929  
Attorney of Record

408 884 1507  
815 572 8299 FAX